Heritage and Philosophy

N. Kean

NMIT: [13522542]

Author Note

This essay was funded by a generous grant from the author's wife and fueled by the spirit of Ardbeg distillery.

Table of Contents

[Abstract 3](#_Toc118466468)

[Heritage and Philosophy 4](#_Toc118466469)

[Heritage 4](#_Toc118466470)

[Philosophy 4](#_Toc118466471)

[Platforms, Libraries, IDEs and code documentation 6](#_Toc118466472)

[Platforms 6](#_Toc118466473)

[Libraries 6](#_Toc118466474)

[IDEs 6](#_Toc118466475)

[Code documentation 6](#_Toc118466476)

[Typing and Sigils/Readability 8](#_Toc118466477)

[Typing 8](#_Toc118466478)

[Sigils/Readability 8](#_Toc118466479)

[Summary/Conclusion 10](#_Toc118466480)

[References 11](#_Toc118466481)

Abstract

The days of programming holy wars have been largely confined to the realms of mythology: vim vs emacs; big-endian vs little-endian; tabs vs spaces. To set a match to the touch paper once again, this essay will compare the Perl and Python programming languages.

Keywords: Perl, Python, Programming, Larry Wall, Guido van Rossum, programming language comparisons.

Heritage and Philosophy

## 

## Heritage

Larry Wall created the Perl language to make report writing and text processing easier (Sheppard, 2000) while Guido van Rossum created the Python language to aid in developing operating system utilities (Severance, 2015). Both developers were filling a perceived gap between the difficulty of the C language and the limitations of the standard UNIX shell tools: bash; sed; awk; etc. (Wall, 1999) (van Rossum, 2009).

The development cycles of both languages follow a strikingly similar path. In the late 1980s, a lone developer identifies a need for a new language and works on it by themself for a while, eventually releasing it as an open-source project on Usenet. Perl in 1987 (Ashton, 1999) and Python in 1989 (Severance, 2015). Van Rossum would eventually be employed by Corporation for National Research Initiatives (CNRI) to focus on Python work (Severance, 2015) while Wall would be hired by O’Reilly Media to further his work on Perl and write books on the language (Wall, 2012).

Like most successful projects, both Perl and Python eventually outgrew their original designers/developers. Today Python is managed by the Python Software Foundation (PSF, 2022), van Rossum having stepped down from his “Benevolent Dictator for Life” role in 2018 (van Rossum, 2018). While Larry Wall still nominally presides over all governance issues for Perl, he now devotes most of his time to the Raku language (previously known as Perl6) (PerlDoc, 2022) and a core “Perl porters” group exists to manage release cycles (not just ports, despite what the name implies). The “Perl porters” group are sub-group of The Perl Foundation (PF, 2022), an American nonprofit organization. Similarly, the Python Software Foundation is an American nonprofit organization that looks after Python issues (PSF, 2022).

## 

## Philosophy

The primary philosophy of Perl can perhaps best be summed up by the phrase “there is more than one way to do it” (Chromatic, 2014). This is in stark contrast to the “Zen of Python” which is “There should be one-- and preferably only one --obvious way to do it.” (Peters, 2004). Perl embraces the notion that it is up to the programmer to code in the way that they find the most efficient, the most readable and the most useful. On the other hand, Python enforces a more proscriptive programming style (van Rossum et al., 2001).

|  |  |
| --- | --- |
| **Perl** | **Python** |
| if true {  print "Hello"  }  print "Hello" if true;  print "Hello"  unless not true;  map { print } "Hello" | if True:  print("Hello") |

The definitive Perl reference is “Programming Perl”, known by the community as “The camel book” (Wall et al., 1991). However, this book places little emphasis on programming style. Larry Wall has stated he does not have strong preferences on Perl code layout, naming conventions, or other elements of style (Sheppard, 2000). Conversely, Python has an official style guide (van Rossum et al., 2001).

Platforms, Libraries, IDEs and code documentation

## Platforms

Both Perl and Python were originally developed on UNIX systems but now have ports for systems as varied as MS Windows, MacOS, FreeBSD, Atari, RISCOS, etc (Fish, 2015) (Stinner, 2022).

## Libraries

A programming language is only as useful as its libraries and both languages have extensive library support. Perl has CPAN (The Comprehensive Perl Archive Network) and Python has the Python Package Index (PyPi) to easily install new libraries and manage dependencies.

|  |  |
| --- | --- |
| **Perl** | **Python** |
| ~ $ cpan install Log4perl | ~ $ pip3 install pysimplegui |

Library archives can be a security hole (Paganini, 2021) as compromising a library in an archive can lead to that compromise being deployed on countless machines worldwide. PyPi alone has 613,000 active users (Purdy, 2022). With the stakes so high, archive maintainers must place a considerable emphasis on security.

The primary advantage of library installers is that they automate dependency management and make updates far easier than having a system maintainer manually keep track of releases. This is important for security updates as they should always be rolled out as quickly as possible.

## IDEs

The right IDE will allow a developer to efficiently produce code for their project. Features such as code completion, version control integration, syntax highlighting and error checking on the fly can greatly increase a programmer's productivity (Okeke, 2022). While vi and emacs were once the only kids on the block, now there are enumerable IDEs to choose from. Python has a distinct advantage over Perl when it comes to IDE choice. Visual Studio Code has been the number-one choice of software developers since 2018 (Stack Overflow, 2018|2021) but its support for Perl is severely lacking. Perl Syntax highlighting is supported out of the box, but debugging and other more advanced features require 3rd party extensions.

The number of developers using Perl has dropped to 2.46% of the professional development community while Python has over 17 times that number at 48.24% (Stack Overflow, 2021). It is not a surprise that those who develop and maintain IDEs give their effort and focus to the languages that are more popular.

## Code documentation

Both Perl and Python have mechanisms for in-line code documentation. Python with the docstring convention and Perl with the Pod (Plain old documentation) format. While Python’s docstring method is primarily a text dump used through the help function, Perl’s Pod format was constructed for export to a number of formats, HTML, man page, etc. As a result of this, the Pod format is much richer, supporting multiple header levels, lists and other markups (Burke, n.d.)

Typing and Sigils/Readability

## Typing

Many battles in the holy wars have been fought over strong vs weak and dynamic vs static type checking. Although Perl and Python are both dynamically typed languages, Perl has dynamic/weak type checking while Python is dynamic/strong. Perl will happily convert variables between types on demand, while Python is fussier.

|  |  |
| --- | --- |
| **Perl** | **Python** |
| ~ $ perl -e 'print "1" + 2'  3 | >>> print("1" + 2)  Traceback (most recent call last):  File "<stdin>", line 1, in <module>  TypeError: can only concatenate str (not "int") to str |

Weak typing results in less verbose code but that comes at the cost of readability (Cunningham, 2012). Python code needs explicit type conversions when operating on variables of different types.

|  |  |
| --- | --- |
| **Perl** | **Python** |
| ~ $ perl -e 'print "1" + 2'  3 | >>> print(int("1") + 2)  3 |

Perl code is easier to write as a developer does not need to worry about forcing type conversions. This comes in especially handy when processing numbers held in text files or from a network stream, as Perl is frequently called upon to do. There is no need to worry about converting the data before processing it. Here is a simple example of adding numbers in a file.

|  |
| --- |
| **Perl** |
| ~ $ cat numbers.csv  1,2,3,4,5,6,7,8  ~ $ perl -e '@\_=split(",",<>); foreach (@\_) {$a+=$\_}; print $a' < numbers.csv  36 |

## Sigils/Readability

Perl denotes scalars, arrays and dictionaries using the sigils $ @ and % while Python doesn’t use sigils at all and the type of a variable is determined by its first use.

|  |  |  |
| --- | --- | --- |
|  | **Perl** | **Python** |
| Scalar | $x = "Something"  $x = $y | x = "Something"  x = y |
| Array/ List | @x = ( 1, 2, 3 ) | x = [1, 2, 3] |
| Hash/ Dictionary | %x = ( a=>1, b=>2 ) | x = dict(a=1, b=2) |
| Subroutine/Function | x = \&a\_sub | x = a\_sub |
| Typeglob (variable aliasing) | \*b = \*a | No equivalent? |

Python code looks cleaner than the equivalent Perl but the downside is that the reader needs to keep track of the type of the variable themselves. This can make debugging harder as even modern IDEs won’t flag the following Python code as problematic, while the Perl code is obviously invalid:

|  |  |
| --- | --- |
| **Perl** | **Python** |
| @x = ("goat");  $y = "rabbit" + @x; | x = ("goat",)  y = "rabbit" + x |

Perl code, especially that which makes extensive use of regular expressions, has the reputation of being “write-only”. Regular expressions are easy to write but can be tricky to understand. The following is an example of particularly messy, yet valid, Perl code (Perl Monks, 2001).

|  |
| --- |
| **Perl** |
| use strict;  $\=$/;  $\_=q&&;  s()/#?=>$":}\~>\\!;/;  $/ =~ s~~s{;};$")!{$"\\:<\_!;~e;  y{%\*-.$'&(#-<}$@-~$s;  {  y(;-?)= r-{)!=s;  y$T-\_$`-|$;  y}{-~}l-\}};  s{!}$Y<$g && redo  }  print; |

You would have to have the brain of a compiler to make sense of that. Admittedly that example was intentionally obfuscated but even “regular” Perl code can still look like line noise and be very hard for a human to parse. Take this simple example: (Cozens, 2000)

|  |
| --- |
| Perl |
| s/\b([tT])eh\b/$1he/g |

Although that is a very easy thing to write, it takes even a seasoned Perl programmer a few seconds of thought to see that it is correcting the common misspelling “Teh” (and “teh”) to “The” (and “the”) in the default variable. The tendency to use regular expressions simply because they are so quick and easy to *write* tends to make understanding uncommented Perl code, much less debugging it, a bit tricky.

Perl programmers make extensive use of regular expressions in their code because of their power and the fact that Perl makes them such an integral part of the language but it is not just regular expressions that lead to the “line-noise” reputation though. The sigils and excessive use of predefined variables contribute to “noisy” code. Who can remember the difference between $\ and $/ when debugging code anyway?

Summary

There is little to be gained by arguing over what language is “better”; They are just different. I came to Perl as a UNIX user who had been stringing together pipelines of grep, awk, etc. I was blown away by the power of Perl’s text-processing abilities. If you have ever tried writing a CGI script in bash, you would have found Perl a godsend too. When I first started learning Python I converted a small Perl program I’d written the week before and was struck by just how much that Python code looked so much cleaner than the Perl version. The shiny new thing may always glisten in our eyes but as the saying goes, “The right tool for the right job”. I will still use Perl for most text-processing tasks and for most other jobs I will probably look first at Python but sometimes all you really need is a grep.

References

Ashton, E. (1999). *The Timeline of Perl and its Culture*. Perl.org.

<https://web.archive.org/web/20130111100906/http:/history.perl.org/PerlTimeline.html>

Burke, S. M. (n.d.) *Plain Old Documentation: format specification and notes* Perl.org

https://perldoc.perl.org/perlpodspec

Chromatic (2014). *Modern Perl (2014 edition)*. Onyx Neon Press

Cozens, S. (2000). *Ten Perl Myths*. Perl.com. https://www.perl.com/pub/2000/01/10PerlMyths.html/

Cunningham, W. (2012). Cunningham & Cunningham, Inc. http://wiki.c2.com/?WeakAndStrongTyping

Fish, S. (2015). *Perl Platforms - The Perl Beginners’ Site*. Perl-Begin.org.

<https://perl-begin.org/platforms/>

Okeke, F. (2022) *The 12 best IDEs for programming*. TechRepublic. https://www.techrepublic.com/article/best-ide-software/

‌PSF (2022). *About the Python Software Foundation*. Python.org. <https://www.python.org/psf/about/>‌

PF (2022). *About the Perl Foundation.*

https://www.perlfoundation.org/about.html

Paganini, P. (2021). *Experts found 11 malicious Python packages in the PyPI repository*. Security Affairs. <https://securityaffairs.co/wordpress/124861/hacking/malicious-pypi-python-packages.html>

*PerlDoc. (n.d.). Various and sundry policies and commitments related to the Perl core.* Perl.org. <https://perldoc.perl.org/perlpolicy>

*PerlDoc. (n.d.). Perl style guide.* Perl.org. <https://perldoc.perl.org/perlstyle>

Peters. T*. (2004). PEP 20 – The Zen of Python* Python.org. <https://peps.python.org/pep-0020/>

Pramanick, S. (2022). History of Python. *Geeks for Geeks*,

<https://www.geeksforgeeks.org/history-of-python/>

Severance, C*.*  (2015). *Guido van Rossum: The Early Years of Python*. Computer.org

<https://www.computer.org/csdl/magazine/co/2015/02/mco2015020007/>

Sheppard, D. (2000) *Beginner’s Introduction to Perl*. Perl.com

Stack Overflow (2018) *Stack Overflow Developer Survey*. Stack Overflow.

<https://insights.stackoverflow.com/survey/2018/>

Stack Overflow (2021) *Stack Overflow Developer Survey*. Stack Overflow. https://insights.stackoverflow.com/survey/2021/#most-popular-technologies-new-collab-tools

Stinner, V. (2022), *Supported platforms and architectures*. Readthedocs.io. https://pythondev.readthedocs.io/platforms.html

‌

van Rossum, G. (2009). *Personal History - part 1, CWI*. Blogspot.com.

http://python-history.blogspot.com/2009/01/personal-history-part-1-cwi.html

van Rossum, G. (2018). *Transfer of Power.* Python-committers mailing list.

https://www.mail-archive.com/python-committers@python.org/msg05616.html

van Rossum, G.; Warsaw, B.; Coghlan, N. (2001*). PEP 8 – Style Guide for Python Code*

Python.org. https://peps.python.org/pep-0008/

Wall, L. (1999). *Perl, the first postmodern computer language,* The Wall Nuthouse, <http://www.wall.org/~larry/pm.html>

Wall. L. (2012). *Larry Wall Live online courses, books, and videos on O’Reilly* Oreilly.com.

<https://www.oreilly.com/pub/au/148>

Wall, L., Christiansen, T., Orwant, J., Foy, D. (1991) *Programming Perl* [O'Reilly Media](https://en.wikipedia.org/wiki/O%27Reilly_Media)‌